Pervasive Parallel Computing

An Historic Opportunity for Innovation in Programming and Architecture

Andrew A. Chien
Vice President of Research
Intel Corporation

Symposium on
Principles and Practice of Parallel Programming
March 16, 2007

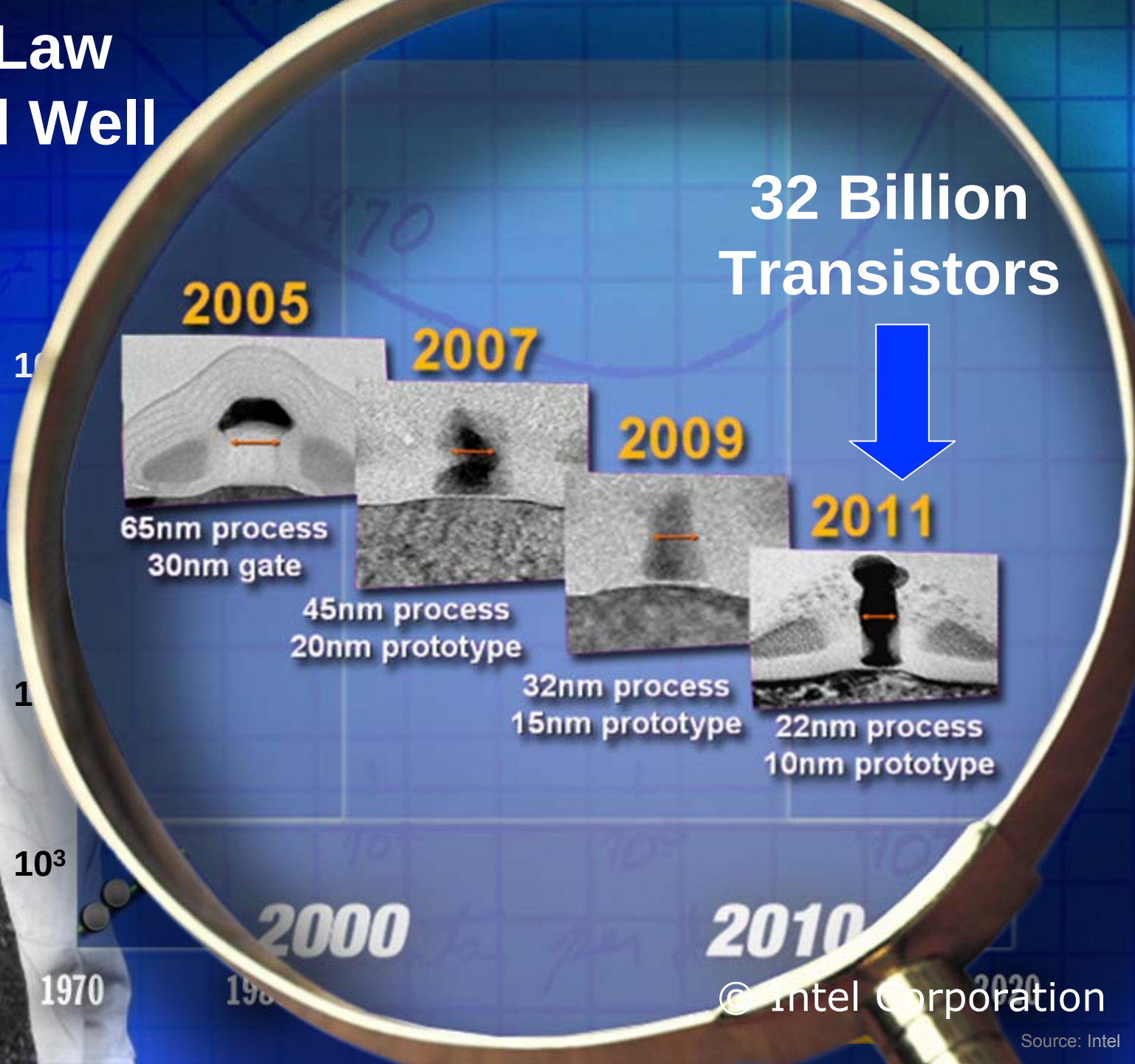
Outline

- Moore's Law and Many Cores
- What Applications Need
- Programming Approaches
- Looking Ahead

- … 4 Big Opportunities in ManyCore…

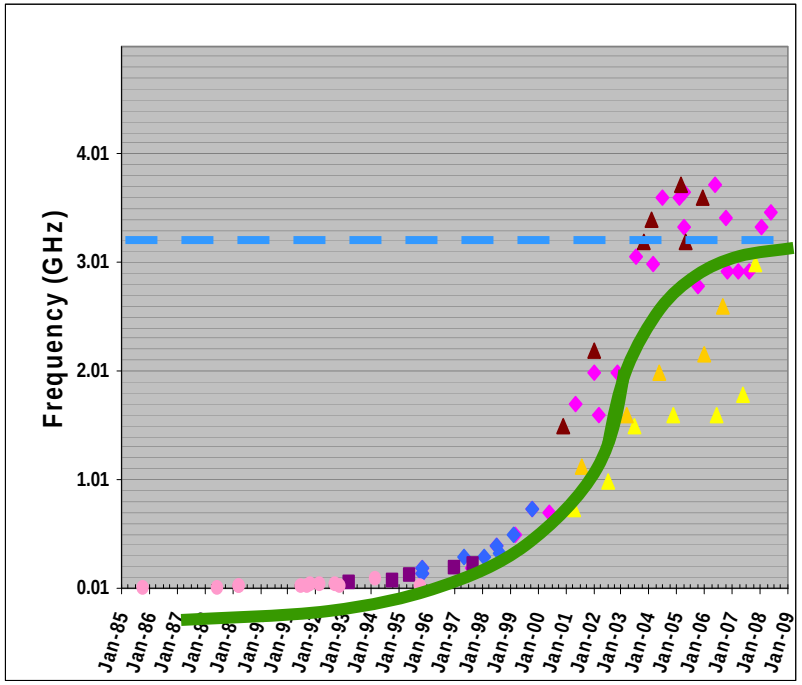
Moore's Law and Many Cores (Terascale)

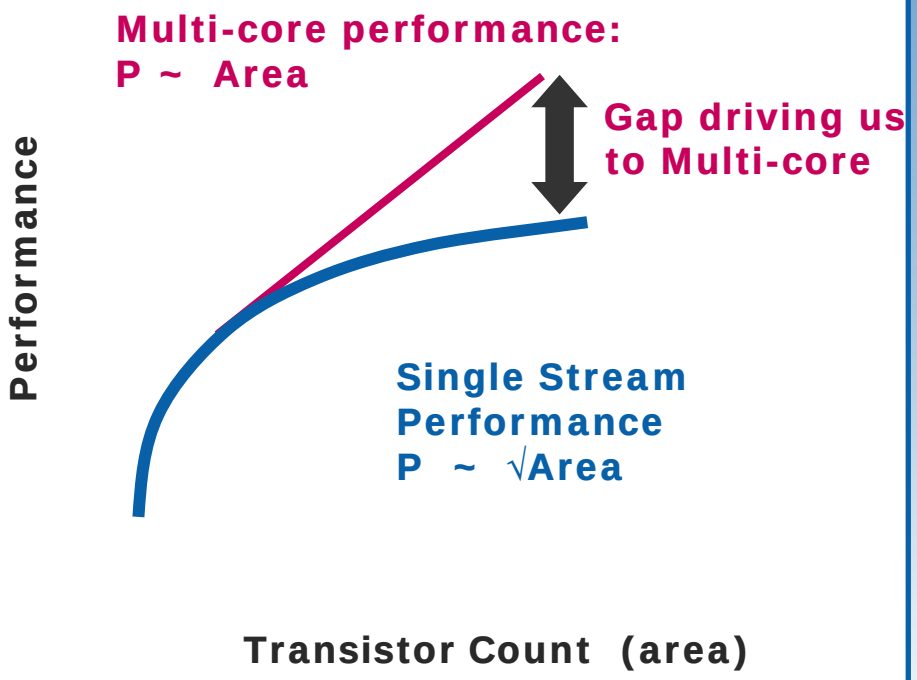
Moore's Law Alive and Well



Source: Intel

Single Core Performance is Stagnant





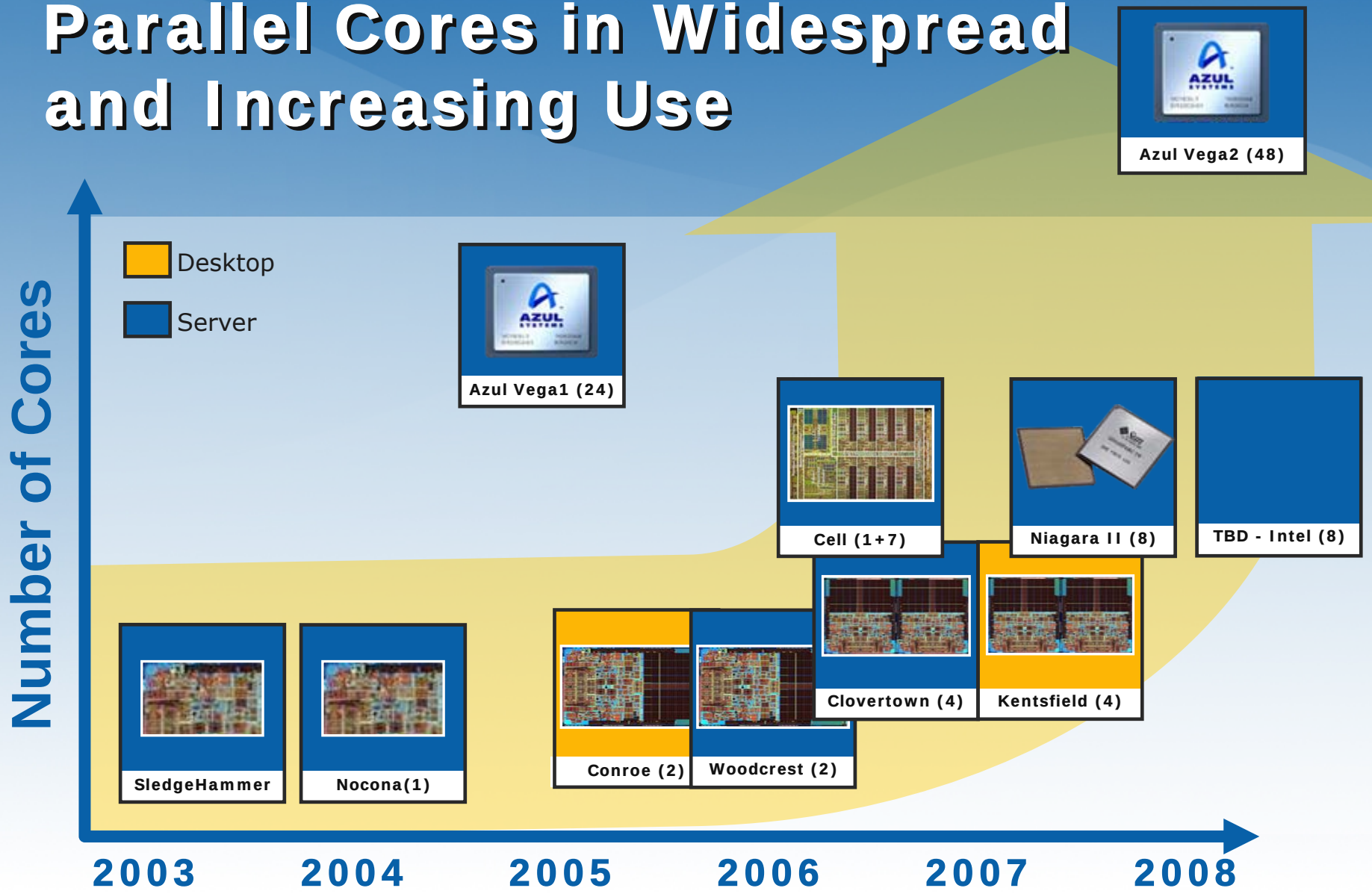
Frequency limited by leakage and power. Transistor counts continue to increase.

Translating Transistors to Performance

- **Si Process**
- **Clock Rate**
- **Micro-architecture (ILP)**
- **Caches**
- Thread Parallelism

- Si Process
- Clock Rate
- Micro-architecture (ILP)
- **Caches**
- **Thread Parallelism**

Parallel Cores in Widespread and Increasing Use



Multi-core is already Mainstream

100%

"Intel to ship roughly 5-6 million of quad-core desktop processors in 2007"

"Sun released its first Niagara-based servers a year ago and currently sells more than $100 million of the systems per quarter"

"We expect Clovertown to represent approximately 28% of Intel servers in 1Q, and 36% in 2Q (2007)"

"Multi-Core chips shipments prevailed over Single-Core offerings… over 50% of Intel's chips were Multi-Core"
[Stacy Smith, Intel]

"…the company was on schedule …for global shipment of 6 million PS3 machines by March 31 (2007)"
[Nanako Kato, Sony spokeswoman]

SONY

“Intel's two new 50-watt [quad core] processors, the Xeon L5320 and the L5310, offer clock speeds of 1.86GHz and 1.6GHz, respectively. Both chips also have a total of 8MB of Level 2 cache and use a 1066MHz FSB.”

- Scott Ferguson, E-week

March 9, 2007

- **With exponentials, the first couple of doublings aren’t too bad, but when the numbers get large, the doublings are huge!**

Where Will Manycore Chips be Used?

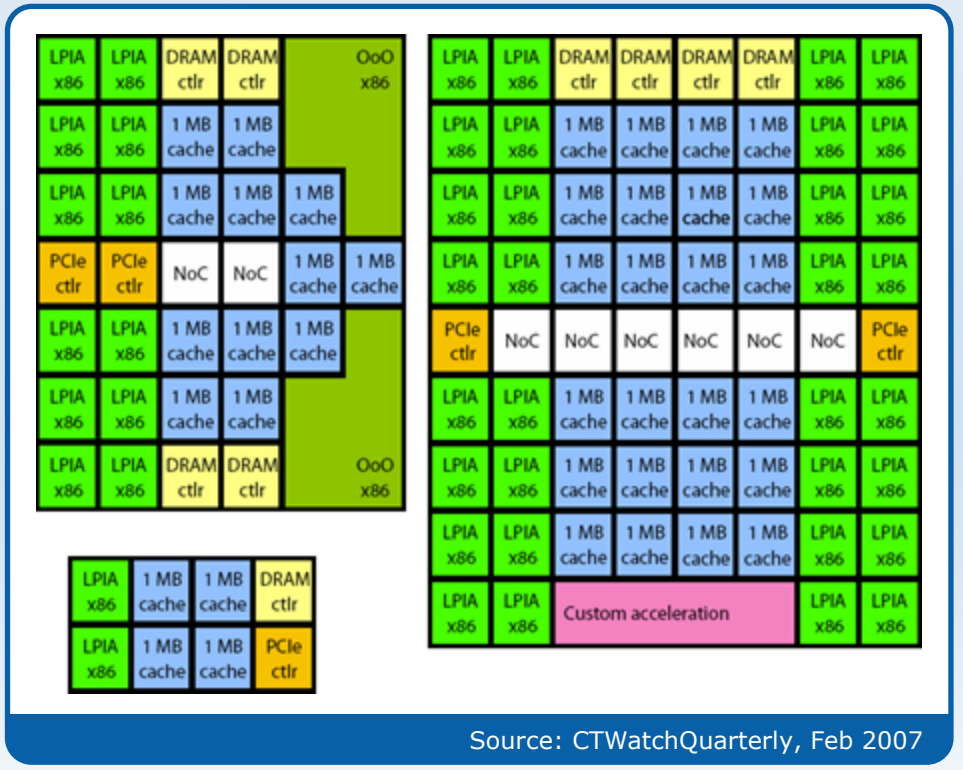
The Obvious

The Edge

The Surprise

Terascale Chip Research Questions

- **Cores**
  - How many? What size?
  - Homogenous, Heterogeneous
  - Programmable, Configurable, Fixed-function
- **Chip-level**
  - Interconnect: Topology, Bandwidth
  - Coordination
  - Management
- **Memory Hierarchy**
  - # of levels, sharing, inclusion
  - Bandwidth, novel Technology
  - Integration/Packaging
- **I/O Bandwidth**
  - Silicon-based photonics
  - Terabit links



Source: CTWatchQuarterly, Feb 2007

Manycore Chips (circa. 2012)?

Opportunity #1: Highly Portable Parallel Software

- Broad range of systems (servers, desktops, laptops, MIDs, smart phones,…) converging to…
  - a single framework with parallelism and a selection of CPU's and specialized elements
  - energy efficiency and performance are core drivers across the board
- => Parallelism moves from niches to widespread pervasive use and permeates all kinds of programming
- => Parallel software can target a very broad range of platforms if standard models of application architecture, expression, and implementation are established.

But, how will we program all of that parallelism?

What do applications need?

Key Software Development Challenges

- Productivity
- Portability
- Performance, Performance Robustness
- Debugging/Test
- Security
- Time to market

⇒ Software Development is Hard!

⇒ Parallelism is critical for performance, but must be achieved in conjunction with all of these requirements…

"Forward scalability": the application continues to get faster on succeeding generations of hardware platforms.

Parallel Programming Non-starters

A Parallel Approach that requires…

- Lower programming productivity
- Major changes in software architecture
- Significant changes in code structure everywhere
- Complicates debugging and test
- Requires maintenance of multiple versions
- Requires deployment diversity (high cost of support)
- Introduces non-modular interactions
  - Major coordinated changes across code
  - Increases correctness dependences
- …

Wanted: Breakthrough Innovations in Parallel Programming

- Parallel Programming Effort ≤ Sequential Programming Effort
- PP Approaches that don't increase programming complexity
  - Productivity, Modularity, Interactions, Performance Tuning, etc.
- PP Approaches that have "forward scalability"
  - To increasing levels of parallelism without reprogramming or retuning
- PP Implementation techniques that enable efficient, high performance, robust parallel performance
  - enable programming at a high level
  - enable performance robustness and portability
- Architecture/Hardware Innovations which Support Parallel Programming
- The Parallel Programming community can address these opportunities!

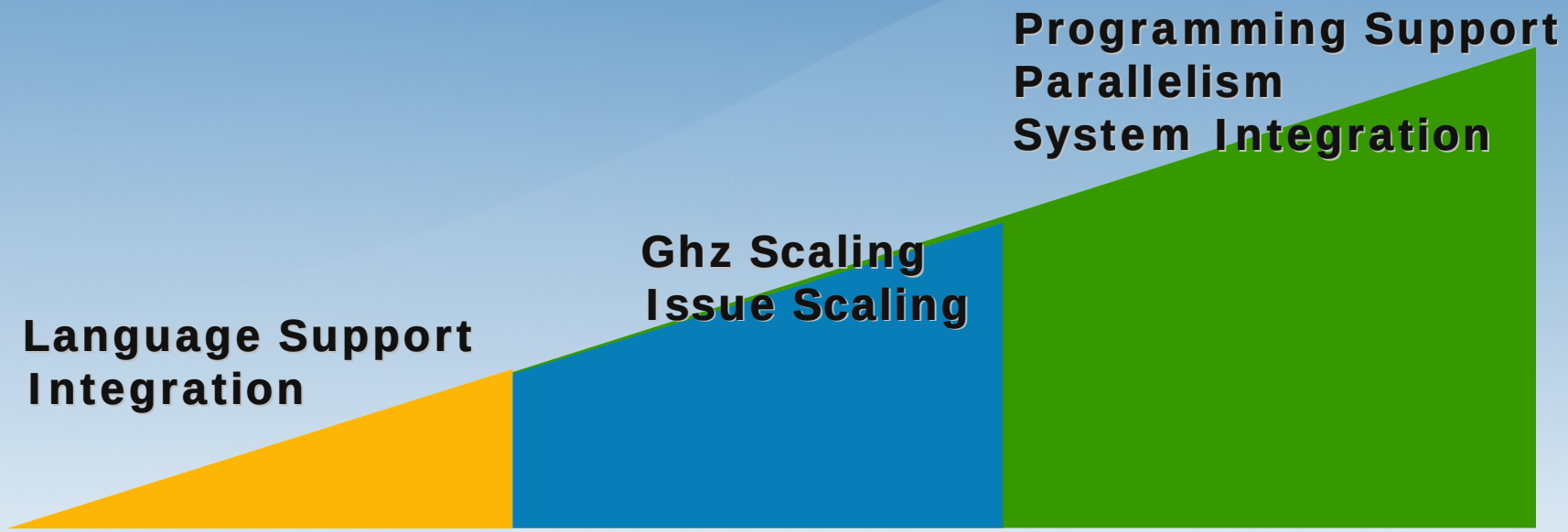
Opportunity #2: Major Architectural Support for Programmability

- Single core growth and aggressive frequency scaling are weakening competitors with other types of architecture innovation

- => Innovations which support functionality – programmability, performance robustness, observability, security, etc. are increasingly possible

- => Don't ask for small incremental changes, be bold and ask for LARGE changes… that make a LARGE difference

New Golden Age of Architectural Support for Programming?



Mid 60's To Mid 80's

Mid 80's To Mid 200x

ManyCore Era

Programming Approaches

Challenges and Approaches

- Exposing the Parallelism
- Managing it for Correctness (and Performance)
- Tuning to Deliver Performance

- Approaches
  - Threading
  - Data Parallel
  - Transactional
  - Declarative / Functional

Threads and Shared Memory

Examples: pThreads, Java, C#, Java, OpenMP

Ex. Matrix Multiply (OpenMP)

```
#pragma omp parallel do private(…) shared(…) setenv OMP NUM THREADS <nthreads>
DO K2 = 1, M, B
  DO J2 = 1, M, B
   DO I = 1, M DO I = 1, M
    DO K = 1, M DO K1 = K2, MIN(K2+B-1,M)
     DO J = 1, M DO J1 = J2, MIN(J2+B-1,M)
       Z(J,I) = Z(J,I) + X(K,I) * Y(J,K) Z(J1,I) = Z(J1,I) + X(K1,I) * Y(J1,K1)
```

Threads and Shared Memory

1. **Who does the management of parallelism and locality?**
   Threads – Programmer; OpenMP – mixed, some runtime help

2. **How does the programmer express parallelism?**
   Programmers describe potential parallelism
   Can express thread interaction and synchronization.

3. **Is the parallelism composable?**
   Requires global reanalysis by programmers

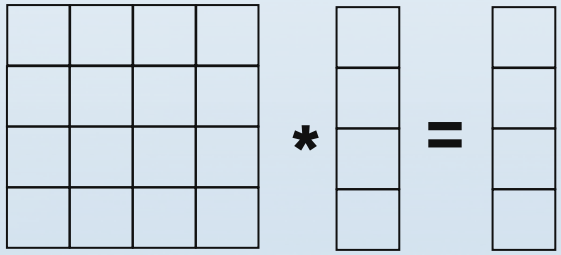
4. **How is performance achieved?**
   Programmer tunes sequential sections.
   OpenMP -- Increasing complex annotation.

Data Parallel Programming

Examples: C*, NESL, Ct, HPF, UPC, Data Parallel C

Ex. Ct matrix vector product

```
CCtVEC<F64>
   SparseMatrixVectorProductCSC(CCtVEC<F64> A,
   CCtVEC<I32> rind, CCtVEC<I32> cols,
   CCtVEC<F64> v) {

   CCtVEC<F64> expv, product, result;

   expv = ctDistribute(v,cols);

    product = A*expv;

    result = ctMultiReduceSum(product,rind);

   return result;

    }
```



Data Parallel Programming

1. **Who does the management of parallelism and locality?**
   Programmers manage alignment (+ data distribution) which is basis for locality.
   System manages degree of parallelism.

2. **How does the programmer express parallelism?**
   Operations on Parallel collections, typically arrays or sets

3. **Is the parallelism composable?**
   Yes for functional/declarative version.
   No, if model uses underlying shared memory (overwrites or overlaps).

4. **How is performance achieved?**
   Manual grain size management, data alignment.

Transactional Memory

Examples: x10, Chapel, T Cilk, TM libraries

Ex. Matrix multiply, Transactional Memory

```
DO K2 = 1, M, B
  DO J2 = 1, M, B
   DO I = 1, M DO I = 1, M
    CALL BEGIN_TRANSACTION(I)
    DO K = 1, M DO K1 = K2, MIN(K2+B-1,M)
     DO J = 1, M DO J1 = J2, MIN(J2+B-1,M)
       Z(J,I) = Z(J,I) + X(K,I) * Y(J,K)  Z(J1,I) = Z(J1,I) + X(K1,I) * Y(J1,K1)
    CALL END_TRANSACTION(I)
```

Transactional Memory

1. **Who does the management of parallelism and locality?**
   Programmer, with some runtime monitoring support
2. **How does the programmer express parallelism?**
   Programmers, with threads
3. **Is the parallelism composable?**
   Yes, in simple cases. Nesting and irreversible operations tricky.
4. **How is performance achieved?**
   Programmer tunes for few aborts.
   Programmer tunes sequential thread sections.

Declarative / Functional Programming

Examples: SQL, ML, Haskell

Ex. Matrix Multiply in Haskell

```
matMult      :: (Ix a, Ix b, Ix c, Num d) =>
                Array (a,b) d -> Array (b,c) d -> Array (a,c) d

matMult x y   =  array resultBounds
                   [((i,j), sum [x!(i,k) * y!(k,j) | k <- range (lj,uj)])
                                | i <- range (li,ui),
                                  j <- range (lj',uj') ]
      where ((li,lj),(ui,uj))   =  bounds x
            ((li',lj'),(ui',uj'))   =  bounds y
            resultBounds
              | (lj,uj)==(li',ui')   =  ((li,lj'),(ui,uj'))
              | otherwise            = error "matMult: incompatible bounds"
```

Declarative / Functional Programming

1. **Who does the management of parallelism and locality?**
   Automatic by runtime system.

2. **How does the programmer express parallelism?**
   Implicitly specified by programmers.

3. **Is the parallelism composable?**
   Yes, no implied state sharing

4. **How is performance achieved?**
   Rewriting algorithms, but difficult to control.
   Data distributions in some cases.

Huge Opportunity: Manycore != SMP on Die

Parameter	SMP	Tera-scale	Improvement
On-die Bandwidth	12 GB/s	~1.2 TB/s	~100X
On-die Latency	400 cycles	20 cycles	~20X

- Less locality sensitive; Efficient sharing
- Runtime techniques more effective for dynamic, irregular data and programs

- Can we do less tuning? And program at a higher level?

Opportunity #3: High Performance, High Level Programming Approaches

- Single chip integration enables closer coupling (cores, caches) and innovation in intercore coordination

- "High Performance without detailed control"
  - High Productivity, High Performance
  - Flexible, modular software

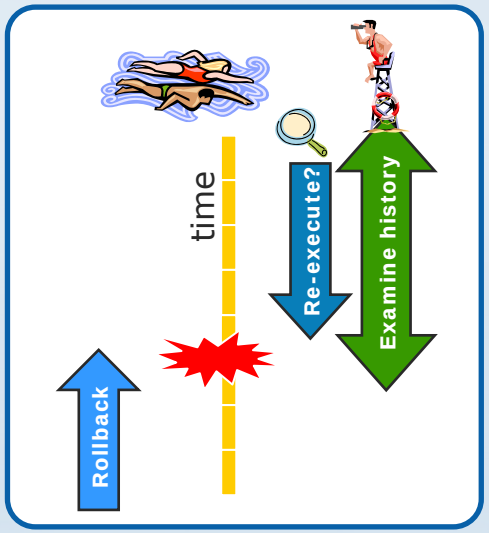
- => Robust performance and "forward scalability"
- => Architecture support for high level programming approaches – optimization and dynamic runtime techniques
  - Isolation and modularity
    - Transactions are just one approach
  - Scheduling and synchronization
  - <your invention here>

Opportunity #4: Use Parallelism to Add New Kinds of Capability and Value

- Additional core and computational capability will be available on-chip, can be exploited at modest additional cost
- => Deploy it to improve the application, software, user, experience
- Traditional: Application level – user interface
  - Visual computing, multi-sensor, large data analysis, machine learning, activity inference
- Interesting:
  - Security – network and platform monitoring
  - Software – health monitoring, invariants, debugging and lifecycle monitoring
  - Dynamic tuning and customization

Using Parallelism to Increase Software Robustness (LBA)

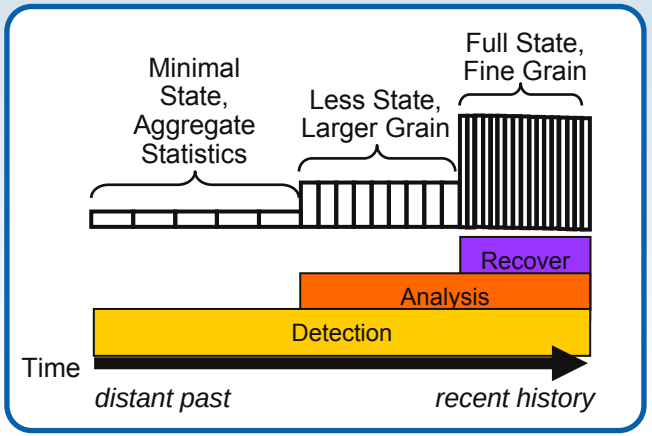
- Programs misbehave too often: bugs, security attacks, hardware faults.
- Runtime tools are too slow to be truly effective.
- The challenges of debugging will increase with multi-core systems.



Research Activities

- Utilize additional performance of multi-core systems for debugging.
- Automatic detection of- and recovery from-software errors. Inspect program's dynamic behavior on a core and use program history to understand failures.
- Efficient dynamic program inspection & rewind via a log that is captured by the hardware, managed by the system and exposed to software

Intel Research Pittsburgh, CMU



Looking Forward

Are All Applications Parallel Applications?

Parallelism Implications for Computing in 2012

- For Education
  - Parallelism front and center
  - An integral part of early introduction and experience with programming
  - An integral part of early algorithms and theory
  - An integral part of software architecture and engineering

- => but this really should happen much sooner!

Parallelism Implications for Computing in 2012

- For Software and Hardware
  - Parallelism in all computing systems
  - Dramatic new capabilities enabled by performance and performance / unit power
  - Established and varied software models for portable parallelism
    - Broad parallel software portability across a wide variety of ManyCore devices
    - Major advances in implementation technologies: tools, languages, environments, runtime
  - Architecture support for Parallel Software and other capabilities
  - Does HPC becomes an integrated niche of the larger ecosystem?

Summary

- Dramatic change and Opportunity around Terascale
  - Moore's law is about parallelism
  - But, Parallelism w/ much closer coupling and higher bandwidth
  - Greater opportunity for integrated parallel support
- Historic Opportunities
  - #1: Highly-portable Parallel Software – servers – small mobiles
  - #2: Major Architecture Support for Programmability
  - #3: High-performance, high-level Parallel Programming
  - #4: Parallelism for new kinds of capability and value
- The Parallel Programming community is the one to address these opportunities!

Questions?